AN EFFICIENT ALGORITHM FOR A CACHE COMPRESSION AND DECOMPRESSION TO IMPROVE SYSTEM MEMORY PERFORMANCE

K.Janaki[1], P.Vijayakumar[2]

[1]Assistant Professor, Department of ECE, Prathyusha Engineering College, Thiruvallur-600025, India.
kr.janaki@gmail.com
[2]Professor, Department of EEE, Karpagam College of Engineering, Coimbatore-641032, India.
vijay.pvk72@gmail.com

Abstract*—Speed is the challenging issue for any electronic component. Memory access time is dependent on speed of the microprocessor. Access time is more in the off-chip memory than on-chip memory. In order to increase the speed, cache memory compression technique is found by microprocessor system designers, as it increases the cache capacity and off-chip bandwidth. Performance of the processor, power consumption and area were assumed in previous work on cache compression. A lossless cache compression algorithm is proposed and designed for high performance processor. This technique allows Parallel compression of multiple words using dictionary mode. Compression ratio is not degraded in the performance.*

Key Terms: Cache memory Compression, data compression, Parallel compression of multiple words.

I. INTRODUCTION

Semiconductor technology rapidly develops and micro architectural developments continue to increase which results in performance gap between processors and memory. Moore's law states that for every two years processor technology doubles in performance and speed.
Modern processors use L1 and L2 as two levels in cache memories to reduce latency and bandwidth. [3]Cache memory compression has been proposed to improve system performance, since effective capacity can be increased by compressing data stored in on-chip caches which reduces cache misses.
When the processor technology increases, speed increases faster because on-chip cache memory hierarchies can store more data in megabyte size. Off-chip memory speed is considerably low compared to processor speed. When the multiprocessor is utilized by system design, it requires more access to memory. Cache compression is used to reduce off-chip communication speed with the processor[5].

The Challenges of Cache memory Compression are:

1. Compression and decompression should be very fast.
2. The hardware should occupy less area and should not increase power consumption.
3. The algorithm should compress even small blocks without losses and should maintain good compression ratio. Compression ratio refers to the ratio between the sizes of the compressed data over uncompressed data.
4. Effective System wide compression ratio must be considered.

Cache compression is one way to improve the effectiveness of cache memories. To reduce latency and bandwidth, cache memories have long been used.

II. RELATED WORK AND CONTRIBUTIONS

i) The **X-Match algorithm:**
This algorithm is mainly based on dictionary entries where in current data is matched with the dictionary entries. 4 byte wide words are entered in the dictionary and many types of matches are possible. The word which do not match with the dictionary are sent separately. X-Match procedure is referred by partial match concept.
The dictionary uses Move to Front (MTF) strategy. The move to front strategy is used to maintain linked list with no duplicate data. When new data is read, MTF is inserted in front of the list. When a duplicate data is read, this data is deleted and inserted again in the beginning.

Though X-Match algorithm is appropriate for compressing main memory, hardware has very large block size which is difficult for compressing the cache lines.

ii) **Frequent pattern Compression (FPC)**
FPC is used to compress cache line by storing frequently appearing word patterns[4]. Cache line is splitted into 32 bit word for compression. Each 32-bit word is encoded as a 3-bit prefix.
If the word matches with any of the patterns given in Table 1, then each word in the cache line is encoded into a compressed format. If the word does not match with any of these patterns, then it is stored in its original 32-bit format i.e. the whole word is stored with the prefix '111'. [11]Cache line compression takes place between L1 and L2 cache during data write in L2 from L1. Decompression takes place when the data is retrieved from L2 to L1.

Table 1.

Prefix	Encoded Pattern	Size of data(bits)
000	Zero run	3
001	4-bit with sign	4
010	8-bits with sign	8
011	Half word with sign	16
100	Half word padded	Half word(non zero)
101	Two half words with sign	16
110	Repeating bytes	8
111	Uncompressed	Original word

Compression can be done easily compared to decompression since prefixes for all words are in series. Prefix is used to find word length of encoded pattern.
Hardware implementation is not possible and so its exact performance, power consumption and area are unknown.

iii) **Restrictive compression technique**
Cache access latency is more focused in this technique and considerably reduced results in increase in the L1 data cache capacity[11]. All words narrow [AWN] is the technique used in this process. Cache block is compared only if all the words in the cache are of narrow width. The AWN technique can be extended by leaving some extra space for a few upper half words in a cache block. In the AWN technique, Least Recently Used policy acts as a replacement policy. In the cache block the byte offset of each word depends on the size of the words that present before it. So to read a word from the block, it will need to recalculate the byte offset. The drawback of this technique is to reduce the cache access latency.

iv) **IBM's Memory Compression**
IBM's Memory Expansion Technology[19] is used for real time main memory compression which effectively double the main memory capacity. Initially it was implemented in the Pinnacle Chip which is single chip memory controller. Data in main memory is compressed using Lempel-Ziv sequential algorithm[2]. This algorithm works by dividing the input data block into sub blocks. The drawback of this algorithm is, it is shown to have negligible performance penalty compared to memory and its contents.

III. GENERAL CACHE COMPRESSION ARCHITECTURE

Lossy versus lossless compression:
A compression algorithm is *lossless* if the decompressed data is identical with the original. Respectively, a compression method is *lossy* if the reconstructed data is only an approximation of the original one.

Performance criteria:
- *Compression efficiency* is the principal element of a compression technique
- The next feature is the *speed* of the compression and decompression process.
- *Distortion* is the major parameter of data compression.

Basics in data compression:
Data compression is mainly consisting of two main factors, *coding* and *modeling*[18]. Coding is based on the selection of the code table. The coding is usually considered as the easy part of the data compression. The modeling of the data for most lossless compression methods are *local* in the way they process the data.

Data representation:
Original data (image) is represented into code pixels and then converted into compressed image data.
Compression ratio considered: Uncompressed data size / compressed data size.

Table 2:

Data (image)size	Recommended Compression ratio
1 kB – 700 kB	4:1
700 kB – 5.5. MB	9:1
5.5 MB – 9.5 MB	16:1
9.5MB – 14.5 MB	25:1
>14.5 MB	36:1

Table 2 represents the recommended compression ratio value for various size of input image data. The typical value considered for this work is 16:1.

In this work, private on-chip L2 caches is examined, because in contrast to a shared L2 cache, the design styles of private L2 cache remain persistent when the number of processor core increases. A system architecture where compression used is shown in Fig. 1. Each processor has L1 and L2 caches[1]. The L2 cache is divided into two regions: an uncompressed (L2) and compressed region (L2C).
For every processor, the size of the uncompressed and compressed region can be determined statically[15]. In most of the cases, the whole L2 cache is compressed due to capacity requirements or uncompressed to minimize access latency. The L1 cache can be used to communicate with the uncompressed region of the L2 cache, which in turn swaps data with the compressed region.

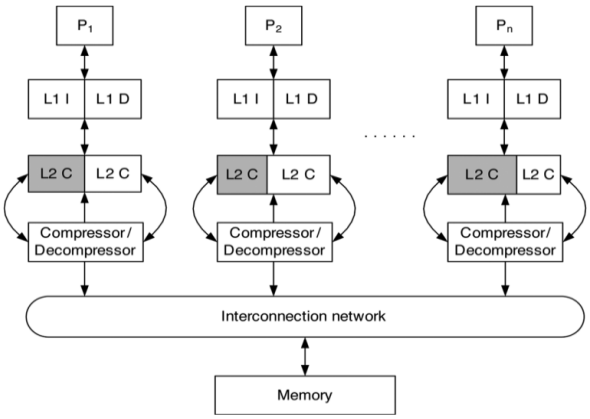


Fig 1. System Architecture where cache compression is used

IV. CACHE MEMORY COMPRESSION ALGORITHM

Design Constraints and Challenges
Several design constraints and challenges to the cache compression[15].
- Cache memory compression requires hardware that can decompress the data in only a few CPU clock cycles.
- Cache compression algorithm must be lossless.

- Block size of the data for cache memory compression is small when compared to file and main memory applications.

Proposed Algorithm Overview

C-Pack algorithm is a lossless compression algorithm particularly for high performance cache compression. Good compression ratio can be obtained when used to compress data commonly found in microprocessor L2 caches[7].

This algorithm achieves compression by,

1) For frequently appearing word, it uses statically decided, compact encodings.

2) For other frequently appearing words, it encodes using dynamically updated dictionary[6]. The dictionary supports partial word matching as well as full word matching.

The patterns and coding techniques used by the algorithm are given in Table-3. The frequently used data is given in pattern column. 'z' denotes 0 byte, 'm' denotes a data byte matched against a dictionary entry, 'x' represents an unmatched byte, 'B' represents a byte and 'b' represents a bit.

Table 3:

Code	Pattern	Output	Length
00	zzzz	(00)	2
01	xxxx	(01)BBBB	34
10	mmmm	(10)bbbb	6
1100	mmxx	(1100)bbbbBB	24
1101	zzzx	(1100)B	12
1110	mmmx	(1110)bbbbB	16

The proposed block diagram for cache compression algorithm is shown in Fig. 2.

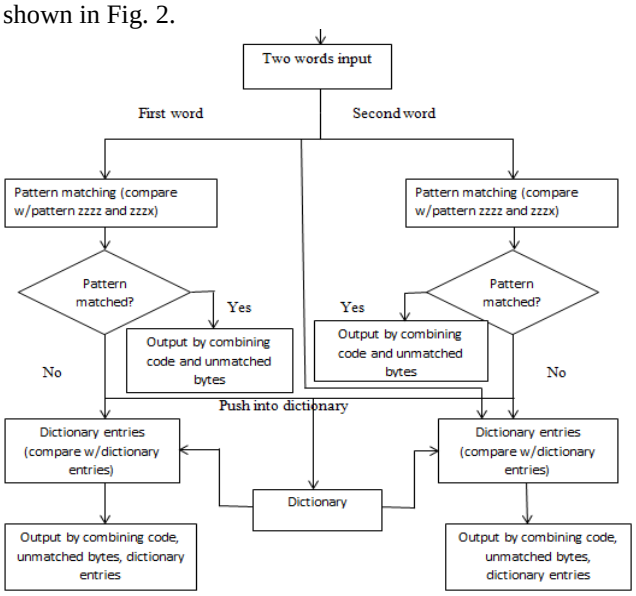


Fig 2(a) C-Pack Compression

Two word input is used per cycle. During first iteration, every word is first compared with specific patterns "zzzz" and "zzzx". If there is a match against patterns, then the output is obtained by combining the relevant code and unmatched bytes. Else the data can be compared with all dictionary entries. The compression output is then generated by combining code, dictionary entry index and unmatched bytes. Data which doesn't match with the patterns is moved into the dictionary.

In decompression, it fetches the compressed words initially and then extracts the codes to analyze the patterns. If the code points a pattern match, the original word is retrieved by combining 0's and unmatched bytes. Else the decompression output is obtained by combining bytes from the input word with bytes from dictionary entries.

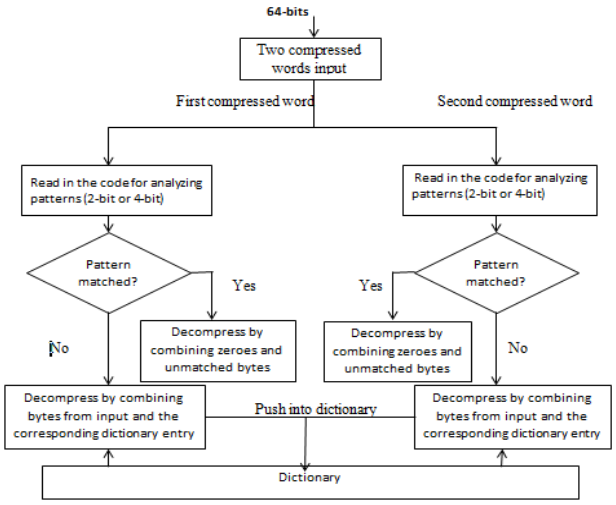


Fig 2 (b). C-Pack decompression

Fig 3 shows how the algorithm works for different input data to produce output.

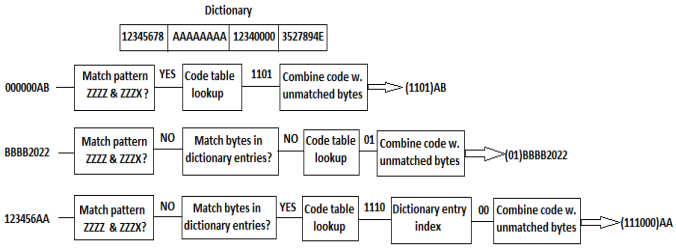


Fig 3: compression with different inputs

The advantage of this algorithm is an input word is compared with multiple patterns and with dictionary entries. This can be permitted for rapid execution with good compression ratio in hardware implementation. To reduce hardware complexity, various design parameters such as dictionary replacement policy and coding scheme can be chosen. In the proposed implementation, two words are processed in parallel per cycle.

V. SIMULATION OUTPUTS

The Compression and decompression outputs based on the algorithm implemented are shown below:

Compression Results

The value for A is 1010 and the value for B is 1011. The input value is 000000AB. The input is compared with "zzzz" and

"zzzx" patterns. If there is a match, it looks up the code and the output is obtained by combining the zeroes (0000), code (1101), and A and B shown in below Fig 4(a).

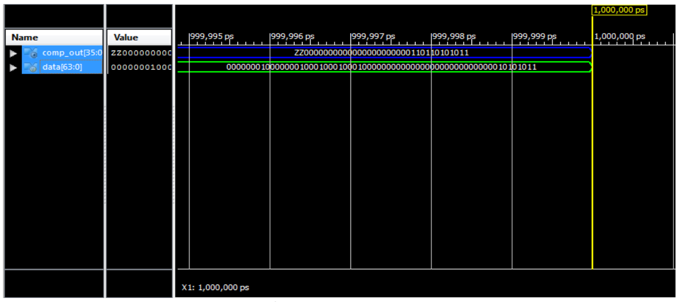


Fig. 4 (a) Compression Output for 000000AB

If there is no pattern match as well as no dictionary match, then the output is obtained by combining the unmatched bytes (zz), code word(01), and the inputs shown in below Fig. 4 (b)

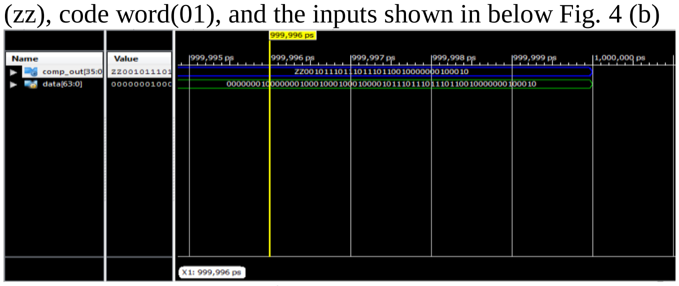


Fig. 4 (b) Compression Output for BBBB2022

Decompression Results

During decompression the original input word is recovered. If the extracted code indicates a pattern match, then the original word is recovered by combining 0's and is shown in Fig 5(a)

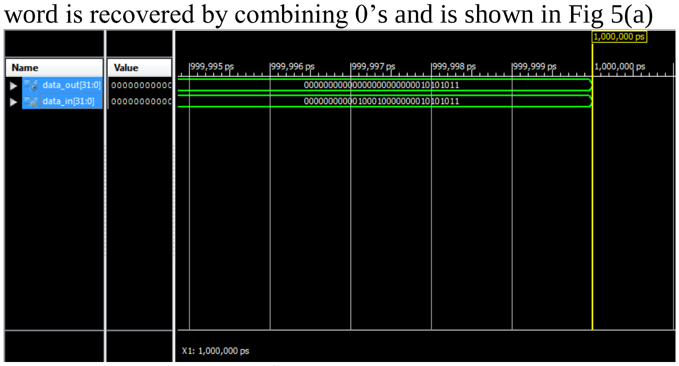


Fig 5(a) Decompression result for (1100)AB

If the code indicates that there is no match with the pattern but there is match with the dictionary entries then the original word is recovered by concatenating the zeroes and unmatched bytes, if any shown in Figure-5(b).

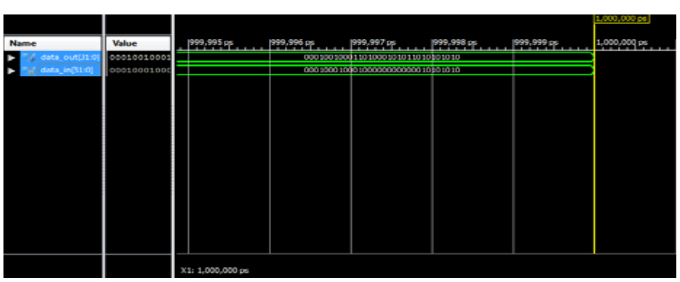


Fig. 5(b). Decompression result for (111000)AA

Compression output if the input data is image:

Final compression output data compression using interpolation is shown in Fig. 6(a). 2048 is total available slot bits for compression. 352 bits compressed output for in the input bits. Group of 9 bits represents compression using left, right and top shift to compress all data.

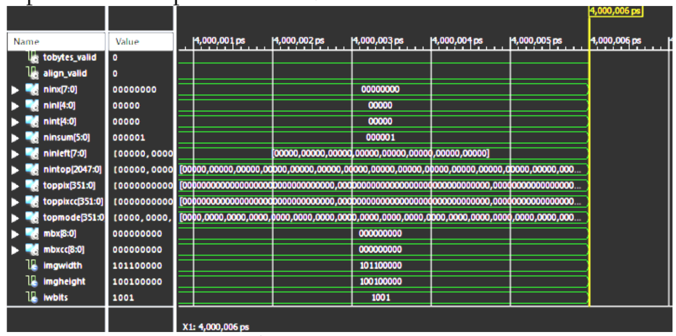
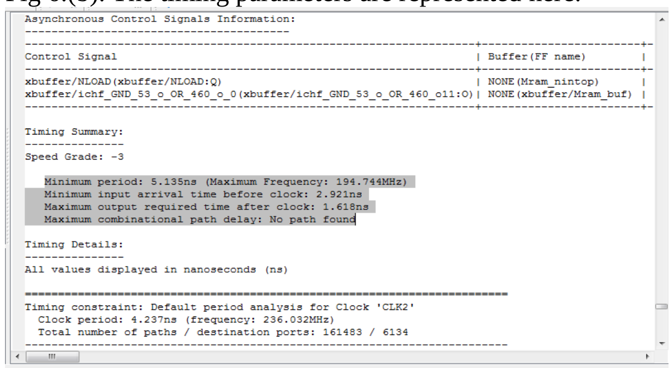
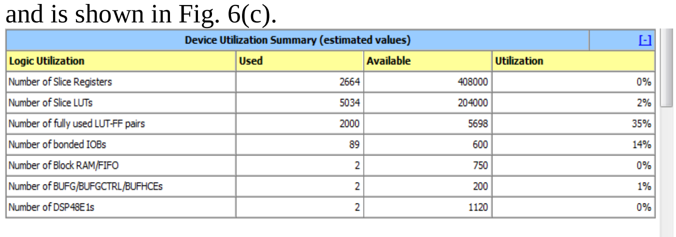


Fig. 6(a) Compressed output for image data

Timing analysis for the compressed output data is shown in Fig 6.(b). The timing parameters are represented here.


```
Asynchronous Control Signals Information:
----------------------------------------
Control Signal                                                    | Buffer(FF name)         |
xbuffer/NLOAD(xbuffer/NLOAD:Q)                                    | NONE(Mram_nintop)       |
xbuffer/ichf_GND_53_o_OR_460_o_0(xbuffer/ichf_GND_53_o_OR_460_o11:O)| NONE(xbuffer/Mram_buf)  |

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 5.135ns (Maximum Frequency: 194.744MHz)
   Minimum input arrival time before clock: 2.921ns
   Maximum output required time after clock: 1.618ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'CLK2'
  Clock period: 4.237ns (frequency: 236.032MHz)
  Total number of paths / destination ports: 161483 / 6134
```
Fig. 6(b) Timing analysis

After the design implementation, design summary is verified and is shown in Fig. 6(c).



Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	2664	408000	0%
Number of Slice LUTs	5034	204000	2%
Number of fully used LUT-FF pairs	2000	5698	35%
Number of bonded IOBs	89	600	14%
Number of Block RAM/FIFO	2	750	0%
Number of BUFG/BUFGCTRL/BUFHCEs	2	200	1%
Number of DSP48E1s	2	1120	0%

Fig. 6(c) Device Utilization Summary

Comparison of Compression ratio

The implemented algorithm is compared with X-Match, FPC, and MXT for mpeg file for cache compression is shown in table 3.

Table 3:

Algorithm	Compression ratio(%)
MXT	75.55
FPC	64.55
X-MATCH	57.97
C-PACK	58.47

CONCLUSION AND FUTURE WORK

The algorithm simulated and synthesized is used for compressing and decompressing the data of 64 or more bits.

Without altering the performance, the data are compressed into the cache in an efficient way. This algorithm produces good compression ratio. Less area is occupied and thus memory latency is decreased so that speed of the system memory performance increases. This algorithm can also be used for high performance lossless data compression applications with or without any modifications.

Future work:

The average or system wide compression ratio can be increased. The algorithm implementation can be lossless. The hardware implementation can be used for high performance lossless data compression applications. The input data can be modeled using statistical linear interpolation and extrapolation techniques in order to process image or video input.

REFERENCES

[1] Sparsh Mittal, "A Survey of Architectural Techniques For Improving Cache Power Efficiency", in Elsevier Sustainable Computing: Informatics and Systems, 2013.

[2] Se-Jun Kwon, Sang-Hoon Kim, Hyeong-Jun Kim, and Jin-Soo Kim, "LZ4m: A Fast Compression Algorithm for In-Memory Data", in IEEE International Conference on Consumer Electronics (ICCE), 2017.

[3] Sparsh Mittal, "A survey of Architectural approaches for data compression in cache and main memory systems", in IEEE transactions on parallel and distributed systems, 2016, pp. 1524-1536.

[4] Yuncheng Guo, Yu Hua, "DFPC-Dynamic Frequent Patten Compression scheme in NVM based main memory", in Design, Automation and Test in Europe Conference, 2018, pp. 1622-1627.

[5]. Esha Choukse, Mattan Erez, "Compress points: An Evaluation Methodology for Compressed memory systems", IEEE Computer Architecture Letters, 2018, pp. 126-129

[6]. David Kaeli, "Dual Dictionary compression for last level cache", IEEE international conference on computer design, 2017, pp. 353-360.

[7] E. G. Hallnor and S. K. Reinhardt, "A compressed memory hierarchy using an indirect index cache," in *Proc. Workshop Memory PerformanceIssues*, 2004, pp. 9–15.

[8] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high-performance processors," in *Proc. Int. Symp. Computer Architecture*, Jun. 2004, pp. 212–223.

[9] J. L. Núñez and S. Jones, "Gbit/s lossless data compression hardware," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 3, pp. 499–510, Jun. 2003.

[10] A. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for 12 caches," Dept. Comp. Scie. , Univ. Wisconsin-Madison, Tech. Rep. 1500, Apr. 2004.

[11] P. Pujara and A. Aggarwal, "Restrictive compression techniques to increase level 1 cache capacity," in *Proc. Int. Conf. Computer Design*, Oct. 2005, pp. 327–333.

[12] L. Yang, H. Lekatsas, and R. P. Dick, "High-performance operating system controlled memory compression," in *Proc. Design AutomationConf.*, Jul. 2006, pp. 701–704.

[13] J.-S. Lee *et al.*, "Design and evaluation of a selective compressed memory system," in *Proc. Int. Conf. Computer Design*, Oct. 1999, pp. 184–191.

[14] N. S. Kim, T. Austin, and T. Mudge, "Low-energy data cache using sign compression and cache line bisection," presented at theWorkshop on Memory Performance Issues, May 2002.

[15] K. S. Yim, J. Kim, and K. Koh, "Performance analysis of on-chip cache and main memory compression systems for high-end parallel computers," in *Proc. Int. Conf. Parallel Distributed Processing TechniquesAppl.*, Jun. 2004, pp. 469–475.

[16] N. R. Mahapatra*et al.*, "A limit study on the potential of compression for improving memory system performance, power consumption, and cost," *J. Instruction-Level Parallelism*, vol. 7, pp. 1–37, Jul. 2005.

[17]. A study material on Data Compression, Google study.

[18] David Salomon, "A Guide to Data Compression Methods", Springer 2001, Chapters 1-2.

[19] R. B. Tremaine, et al., "Pinnacle: IBM MXT in a memory controller chip," in Proc. Int. Symp. Microarchitecture, Apr. 2001.

[20] K.Janaki, P.Vijayakumar, "Proposed Cache Compression Algorithm for Microprocessors to Improve System Memory Performance", in International Journal of Applied Engineering Research, 2015, pp.1001-1007.

[21] K.Janaki, P.Vijayakumar, "A Novel approach for a high performance lossless cache Compression algorithm", in ARPN Journal of Engineering and Applied Sciences, 2015, Vol 10, No 7.

[22] K.Janaki, P.Vijayakumar, "Adaptive Lossless Cache Compression Algorithm", in Special Issue on IEEE Sponsored International Conference on Intelligent Systems and Control (ISCO'15), Vol No II.